5K

УДК 004.832

В.В. Краснопрошин, А.В. Карканица
Белорусский государственный университет, г. Минск, Беларусь
Гродненский государственный университет имени Янки Купалы, г. Гродно, Беларусь
krasnoproshin@bsu.by, karkanica@gmail.com

Алгоритмы модификации деревьев для построения динамических предметных областей

В статье рассматривается задача построения модели динамической многоуровневой предметной области для решения сложно структурированных задач. Предметная область такого класса задач формируется из различных информационных источников, распределенных в глобальной среде. Предлагается структура данных и алгоритмы модификации деревьев для построения динамических предметных областей.

Введение

При решении сложных задач возникает потребность в распределении работ между исполнителями с последующим объединением полученных результатов. На этапе декомпозиции центр может определить основные подзадачи, но ввиду естественных ограничений (компетенция, ресурсы, время) не способен найти их решение. Поэтому возникает проблема распределения подзадач между различными исполнителями, а после получения частных результатов – проблема их согласования и интеграции [1]. Решение проблемы распределения подзадач состоит из четырех этапов:

1) декомпозиция центром задачи S на подзадачи (S_{11}, S_{12}, S_{21}, …);
2) назначение подзадач исполнителям (E_1; E_2; E_3; …);
3) решение подзадач исполнителями;
4) синтез общего решения из частных результатов.

Декомпозиция заключается в генерации множества подзадач, которые потенциально могут быть переданы другим исполнителям. Решение подзадачи исполнителем E_i, которому она передана, может состоять в полностью самостоятельном решении. Однако этот исполнитель также может решить проблему распределения, т.е. в свою очередь произвести декомпозицию подзадачи и привлечь других исполнителей. Процесс разбиения задачи на подзадачи может состоять из нескольких уровней.

Синтез результата производится центром, который инициировал распределение задачи. Он собирает частные результаты своих «помощников» и формирует общий результат. Процесс формирования зависит от того, как исходная задача была разделена на подзадачи. Если декомпозиция, как отмечено выше, была многоуровневой, то и синтез результата будет многоуровневым.

Таким образом, онтология предметной области задачи S строится в процессе совместной деятельности групп экспертов. Автоматизация этого процесса требует создания программной системы, которая должна поддерживать согласованную работу географически распределенных экспертов по формированию онтологии предметной области. В [2] показано, что соответствующая модель предметной области может быть описана следующим образом:

$$modY=(Entity, Relation, ERD \mid P, Z) \quad (1)$$

Первые три компонента модели (1) представляют древовидный ациклический граф, вершиной которого является задача **S**, узлы определяют иерархию подзадач, дуги – уровень их вложенности. Терминальные узлы графа представлены фреймами **P** (содержательная составляющая подзадачи) и **Z** (технолого-образовательная составляющая).

Предлагается алгоритм построения модели (1), состоящий из двух этапов. На первом из них необходимо построить статическую часть модели, представляющую иерархию задач целевой задачи. На втором – динамическую часть, подключая к терминальным узлам фреймы **P** и **Z**. В результате формируется полная динамическая модель ПрО, представленная ациклическим графом с терминальными узлами, представленными фреймами.

Свойство динамичности модели (1) обусловлено тем, что топология подзадач может меняться по мере подключения новых исполнителей. Это, в свою очередь, ведет к соответствующей модификации графа целевой задачи.

Целью данной работы является разработка алгоритмического аппарата, позволяющего выполнять построение иерархического древовидного графа целевой задачи и его модификацию. Рассмотрим возможности реализации этого этапа на основе аппарата теории графов.

Основные понятия и определения

Пусть **G(V, E)** – древовидный иерархический граф, представленный множеством вершин **E** и множеством ребер **V**. Граф **G** будем называть динамическим, если за промежуток времени **t** возможен переход графа из состояния $\mathbf{G^1(V^1, E^1)}$ в состояние $\mathbf{G^2(V^2, E^2)}$, причем множества $\mathbf{(V^1, V^2)}$ и $\mathbf{(E^1, E^2)}$ соответственно не совпадают.

Модификацией графа **G** назовем процесс перехода графа из состояния $\mathbf{G^1}$ в момент времени $\mathbf{t_1}$ в состояние $\mathbf{G^2}$ на момент времени $\mathbf{t_2}$, который может быть вызван выполнением некоторой последовательности двух типов операций на графе: слияние (**link**) и разбиение (**cut**). В дальнейшем древовидный иерархический граф будем называть деревом.

Операцию **слияния (link(v, w))** определим как операцию, результатом выполнения которой является добавление дуги **edge(v, w)** в дерево **G**. Здесь **v** – вершина дерева **G**, **w** – новая вершина. В результате выполнения операции слияния вершина **w** становится потомком вершины **v** (рис. 1).

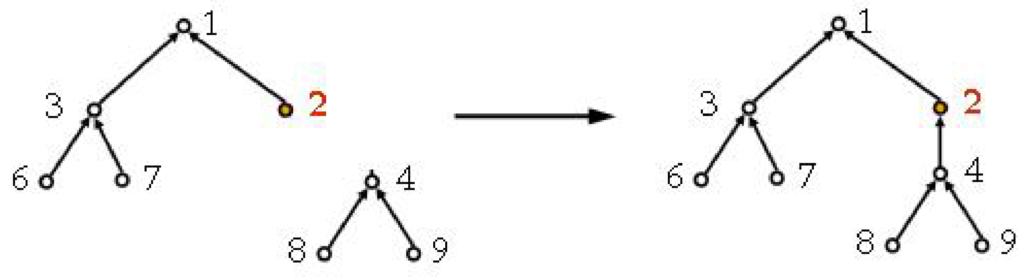


Рисунок 1 – Результат выполнения операции *link(4, 2)*

Операцию **разбиения** дерева **G (cut(v, w)),** определим как операцию, результатом выполнения которой является разбиение дерева G, содержащего дугу **(v, w)** на два дерева, удалением дуги **(v, w).** Здесь **v** и **w** – смежные вершины одного дерева (рис. 2).

Путь в дереве G определим как последовательность вершин $\mathbf{v_i \in V}$, таких, что две любые последовательные вершины соединены хотя бы одной дугой из множества E.

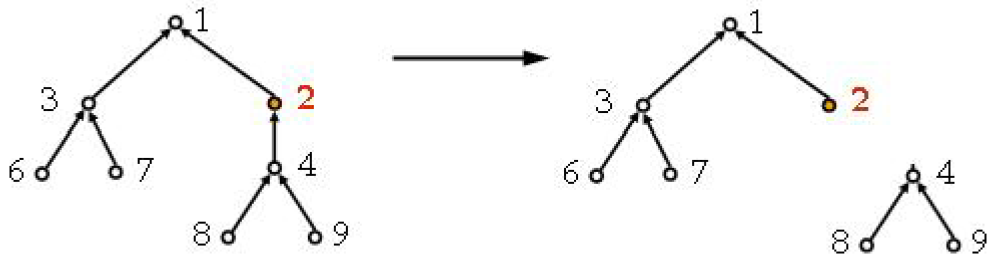


Рисунок 2 – Результат выполнения операции *cut(4, 2)*

Анализ результата выполнения операций *link* и *cut* (рис. 2, 3) позволяет сделать вывод о том, что операция слияния вершин **v** и **w** фактически приводит к формированию пути от корня дерева к новой вершине (путь (1, 2, 4, 9) на рис. 2). Напротив, операция **cut** разбивает путь между вершинами **v** и **w** на два пути: от корня до вершины **v** (путь (1, 2) на рис. 2) и от вершины **w** до терминальной вершины (путь (4, 9) на рис. 2). Очевидно, что выполнение каждой из введенных операций в отдельности, а также выполнение некоторой последовательности таких операций, приводит к модификации исходного дерева **G.** А задачу модификации дерева можно свести к задаче модификации путей дерева **G**.

Введенные операции можно считать элементарными, однако их формальное описание может быть представлено только в терминах элементарных операций над путями и операций, допустимых на множестве вершин и множестве ребер динамического дерева.

На множестве вершин и множестве ребер дерева **G** введем следующий набор допустимых операций и определим правила их выполнения.

Таблица 1 – Допустимые операции на графе

№	Операция	Параметры	Результат выполнения операции
1	*parent*	*vertex v*	Результатом выполнения операции является ссылка на предка вершины *v*. Если вершина *v* не имеет предка, то она является корнем дерева и результатом выполнения операции является специальное значение *null* (пусто).
2	*root*	*vertex v*	Операция возвращает ссылку на корень дерева, содержащего вершину *v*.
3	*cost*	*vertex v*	Операция возвращает вес ребра *edge(v,parent(v))* и является допустимой для вершины, не являющейся корнем дерева.
4	*mincost*	*vertex v*	Операция возвращает ссылку на вершину, ближайшую к *root(v),* такую что ребро *edge(w,parent(w))* имеет минимальный вес среди ребер на пути от *v* к корню *root(v).* Операция допустима, если *v* не является корнем дерева.
5	*update*	*vertex v, real x*	Операция обновления весов всех ребер на пути от вершины *v* к корню *root(v)* добавлением значения *x* к весу каждого ребра.
6	*path*	*vertex v*	Возвращает путь, содержащий вершину *v*.
7	*head*	*path p*	Возвращает начальную (первую) вершину пути *p*.
8	*tail*	*path p*	Возвращает хвост (последнюю вершину) пути *p*.
9	*after*	*vertex v*	Возвращает вершину, стоящую за вершиной *v* в пути *path(v*). Если вершина *v* – это концевая вершина пути *path(v)*, то возвращает специальное значение *null*.

Продолжение табл. 1

№	Операция	Параметры	Результат выполнения операции
10	*pupdate*	*path p, real x*	Добавляет вес *x* к весу каждого ребра пути *p*.
	concatenate	*path p, q, real x*	Объединяет пути *p* и *q* добавлением ребра *edge(tail(p), head(q))*. Возвращает полученный путь.
11	*split*	*vertex v*	Разбивает путь *path(v)*, удалением из него ребер, инцидентных *v*. Возвращает список *[p, q]*, где *p* – это часть пути, включающая все вершины от *head(puth(v))* до *before(v)*, *q* – это часть пути, содержащая все вершины из *after(v)* до *tail(puth(v))*.

Таким образом, для решения задачи построения графа целевой задачи и последующей его модификации требуется разработать алгоритмы выполнения операций слияния и разбиения. Покажем далее, что введенная выше совокупность операций на дереве *G* является необходимой и достаточной для формального описания и последующей реализации алгоритмов модификации. Для этого разработаем структуру данных для представления динамического дерева и опишем алгоритмы его модификации в терминах введенных операций.

Структура данных для представления динамического дерева

Пусть на множестве вершин дерева *G* определен набор допустимых операций, обозначенных выше. Требуется разработать структуру данных для представления динамического дерева, обеспечивающую эффективную реализацию процедур разбиения (дефрагментации) и слияния (интеграция фрагментов).

Для решения поставленной задачи предлагается модификация динамической структуры, разработанной Тарьяном и Слейтором [4]. Выбор обоснован тем, что предлагаемая структура обеспечивает амортизационную сложность выполнения операций на дереве – $O(\log n)$, что является важным с точки зрения оценки временной эффективности алгоритмов.

Введем на множестве ребер *E* дерева *G* два типа ребер (сплошные и пунктирные) по следующему правилу: из каждой вершины выходит не более одного сплошного ребра. В соответствии с этим все пути в графе также можно разделить на два типа: сплошной путь и разорванный путь (рис. 3).

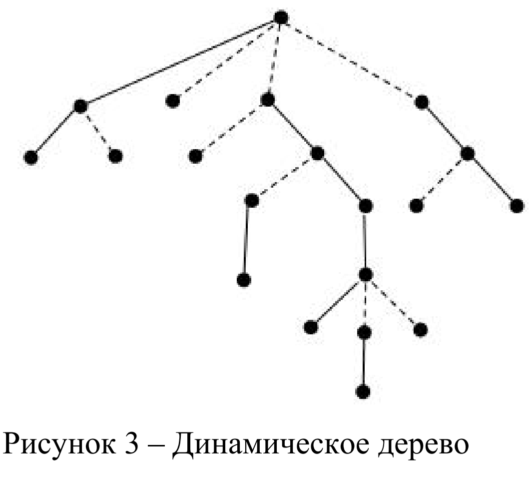


Рисунок 3 – Динамическое дерево

Тогда для каждой вершины *v*, которая является конечной вершиной сплошного пути, будем хранить указатель на ее родителя, из которого выходит пунктирное ребро. Как было показано выше, задачу модификации динамического графа можно свести к задаче модификации путей. В соответствии с предложенной структурой для представления динамического дерева, модификация пути может означать удлинение сплошного пути *p* и построение сплошного пути с начальной вершиной *v* и конечной вершиной *parent(v)*. Эти преобразования опишем двумя типами операций:

1) **splice(path p)** – удлиняет сплошной путь, преобразовывая пунктирные ребра, заканчивающиеся в хвосте пути *p*, в сплошные (рис. 4). Возвращает полученный путь.

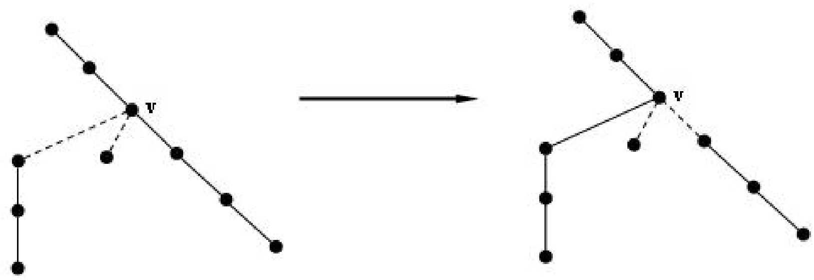


Рисунок 4 – Результат выполнения операции splice

2) **expose(vertex v) –** создает сплошной путь с начальной вершиной *v* и конечной вершиной *parent(v)*, преобразовывая пунктирные ребра в сплошные на всем пути от вершины *v* к *root(v)*. Возвращает результирующий сплошной путь (рис. 5).

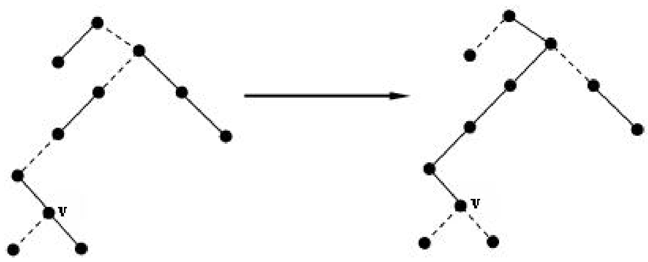


Рисунок 5 – Результат выполнения операции expose

Опишем алгоритмы выполнения операций *splice* и *expose*, используя запись на псевдокоде в нотации Дейкстры.

1. Алгоритм выполнения операции *splice*: на входе алгоритма – разорванный путь *p*, содержащий пунктирные ребра; на выходе – сплошной путь.

```
function splice(path p):
vertex u; path q, r; real x, y,
u := dparent(tail(p));
[q, r, x, y ] := split(v);
If
 q < > null dpurent(tail(q)):=v;
      p := concatenate(p,puth(v), dcost(tail(p)));

if r = null return p;
  r < > null concatenate(p, r, y);
fi
end splice;
```

2. Алгоритм выполнения операции *expose*: на входе алгоритма – вершина *v*, на выходе – сплошной путь от вершины *v* к *root(v)*.

```
function expose(vertex u);
path p, q, r; real x, Y;
[q, r, x, y ] := split(u);
if q < > null dparent(tail(q)):=v; fi;

if r=null p :=puth(v);
  r< >null p:=concatenate(path(v),r,y)
fi;

do dpurent(tail(p)) < > null p := splice(p) od;
return p;
end expose;
```

Алгоритмы модификации динамического дерева

Представленная реализация алгоритмов модификации путей в дереве *G*, а также приведенный набор допустимых операций позволяют сформулировать в терминах этих операций необходимые алгоритмы для построения и модификации динамического дерева: определение родителя вершины *v*, поиск корня дерева, слияние деревьев и разбиение (декомпозиция) дерева.

1. Поиск родителя вершины *v*: на входе алгоритма – вершина дерева *v*, на выходе – непосредственный родитель вершины *v*.

```
function parent(vertex v);
 if v = tail(path(v)) return dparent(v);
 if v< > tail(path(v)) return after(v);
end parent;
```

2. Определение корня дерева: на входе алгоритма – вершина *v*, на выходе – корень дерева, которому принадлежит вершина *v*

```
function root(vertex v);
 return tail(expose(v))
end root;
```

3. Алгоритм слияния деревьев: на входе алгоритма – вершины *v* и *w*, на выходе – дерево, полученное слиянием двух деревьев за счет добавления дуги (*v*,*w*).

```
procedure link(vertex v, w, real x);
      concatenate(path(v), expose(w), x)
end link;
```

4. Алгоритм разбиения (декомпозиции) дерева: на входе алгоритма – вершина *v*: на выходе – два фрагмента исходного дерева, полученные удалением дуги (*v*, *after*(*v*)).

```
function cut(vertex v);
path p, q; real x, Y;
expose(v);
[p,q,x,y] := split(v);
dpurent(v) := null;
return y
end cut;
```

Выводы

В работе рассмотрен класс задач, решение которых формируется в результате совместной деятельности групп экспертов (исполнителей). Для решения задачи построения графа целевой задачи и последующей его модификации разработан понятийный и алгоритмический аппарат. Предложена структура данных для представления динамического дерева, введен набор допустимых операций и разработаны алгоритмы модификации дерева, описанные в терминах допустимых операций. Представленные алгоритмы могут быть использованы для создания программной системы, обеспечивающей согласованную работу территориально распределенных групп экспертов по формированию предметной области сложно структурированных задач.

Литература

1. Виссия Х. Технология выполнения IT-проектов коллективами распределенных исполнителей / Х. Виссия, В.В. Краснопрошин, А.Н. Вальвачев // Искусственный интеллект. – 2008. – № 3. – С. 63-69.
2. Карканица А.В. Онтологический подход к построению моделей динамических предметных областей / А.В. Карканица // Вестник Гродненского государственного университета имени Янки Купалы. Серия 2. – 2010. – № 1(92). – С. 92-97.
3. Краснопрошин В.В. Интеграция распределенных экспертных знаний: проблемы и решения / В.В. Краснопрошин, Г. Шаках, А.Н. Вальвачев // Информатика. – Минск, 2004. – № 1. – С. 45-53.
4. Daniel D. Sleator. A Data Structure for Dynamic Trees / Daniel D. Sleator, Robert E. Tarjan // Journal of Computer and System Sciences. – 1983. – № 3. – P. 363-391.

В.В. Краснопрошин, А.В. Карканіца

Алгоритми модифікації дерев для побудови динамічних предметних областей

У статті розглядається задача побудови моделі динамічної багаторівневої предметної області для розв’язання складно структурованих задач. Предметна область такого класу задач формується з різних інформаційних джерел, розподілених у глобальному середовищі. Пропонується структура даних й алгоритми модифікації дерев для побудови динамічних предметних областей.

V.V. Krasnoproshin, A.V. Karkanitsa

Algorithms Modification of Trees to Create a Dynamic Subject Domain

The paper describes a problem of constructing model of dynamic multi-level subject domain for tasks with complicated structure. The subject domain of these tasks has been formed from a variety of information sources, distributed in the global environment. A data structure and algorithms modification of trees to create a dynamic subject domain are presented.

Статья поступила в редакцию 01.07.2010.